A Linear Time Approximation Algorithm for Weighted Matchings in Graphs[⋆]

Doratha E. Drake and Stefan Hougardy

Institut für Informatik, Humboldt-Universität zu Berlin, 10099 Berlin, Germany
{drake,hougardy}@informatik.hu-berlin.de

December 2003

Abstract. Approximation algorithms have so far mainly been studied for problems that are not known to have polynomial time algorithms for solving them exactly. Here we propose an approximation algorithm for the weighted matching problem in graphs which can be solved in polynomial time. The weighted matching problem is to find a matching in an edge weighted graph that has maximum weight. The first polynomial time algorithm for this problem was given by Edmonds in 1965. The fastest known algorithm for the weighted matching problem has a running time of $O(nm+n^2 \log n)$. Many real world problems require graphs of such large size that this running time is too costly. Therefore there is considerable need for faster approximation algorithms for the weighted matching problem. We present a linear time approximation algorithm for the weighted matching problem with a performance ratio arbitrarily close to $\frac{2}{3}$. This improves the previously best performance ratio of $\frac{1}{2}$. Our algorithm is not only of theoretical interest but because it is easy to implement and the constants involved are quite small it is also useful in practice.

Keywords: *Maximum Weight Matching, Approximation Algorithm, Graph Algorithm*

1 Introduction

A *matching* M in a graph $G = (V, E)$ is a subset of the edges of G such that no two edges in M are incident to the same vertex. In a graph $G = (V, E)$ with edge weights given by a function $w : E \to \mathbb{R}^+$ the *weight of a matching* M is defined as $w(M) := \sum_{e \in M} w(e)$. The *weighted matching problem* is to find a matching in G that has maximum weight. The first polynomial time algorithm for the weighted matching problem was given by Edmonds [7] in 1965. A straightforward implementation of this algorithm requires a running time of $O(n^2 m)$, where n and m denote the number of vertices and edges in the graph. Lawler [15] and Gabow [9] improved the running time to $O(n^3)$. Galil, Micali, and Gabow [12] presented an implementation of Edmond's algorithm with a running time of $O(nm \log n)$. This was improved by Gabow, Galil, and Spencer [11] to a running time of $O(nm \log \log \log n + n^2 \log n)$. The fastest known algorithm to date for solving the

[⋆] supported by DFG research grant 296/6-3, supported in part by DFG Research Center *Mathematics for key technologies*, a preliminary version appeared in APPROX03 [6] weighted matching problem in general graphs is due to Gabow [10] and has a running time of $O(nm + n^2 \log n)$.

In some special cases faster algorithms for the weighted matching problem are known. Under the assumption that all edge weights are integers in the range $[1 \ldots N]$ Gabow and Tarjan [13] presented an algorithm with running time $O(\sqrt{n \cdot \alpha(m,n) \log n} \cdot m \log(Nn))$, where α is the inverse of Ackermann's function [24]. In the case that all edge weights are the same, the fastest known algorithm has a running time of $O(\sqrt{n}m)$ and is due to Micali and Vazirani [21, 25]. For planar graphs Lipton and Tarjan [17] have shown that with the help of the Planar Separator Theorem [16] the weighted matching problem can be solved in $O(n^{3/2} \log n)$.

Together with the research on improving the worst case running time of Edmond's algorithm for the weighted matching problem there has been a parallel line of research concerned with the implementations of these algorithms. Implementations of Edmond's algorithm that turn out to be efficient in practice usually not only require the use of sophisticated data structures but also need additional new ideas to lower the running time in practice. During the last 35 years many different implementations of Edmond's weighted matching algorithm have been presented. See [3] for a good survey on these. Currently the fastest implementations of Edmond's algorithm are due to Cook and Rohe [3] and to Mehlhorn and Schäfer [20].

Many real world problems require graphs of such large size that the running time of the fastest available weighted matching algorithm is too costly. Examples of such problems are the refinement of FEM nets [18], the partitioning problem in VLSI-Design [19], and the gossiping problem in telecommunications [2]. There also exist applications were the weighted matching problem has to be solved extremely often on only moderately large graphs. An example of such an application is the virtual screening of protein databases containing the three dimensional structure of the proteins [8]. The graphs appearing in such applications have only about 10,000 edges, but the weighted matching problem has to be solved more than 100,000,000 times for a complete database scan.

Therefore, there is considerable need for *approximation algorithms* for the weighted matching problem that are very fast, and that nevertheless produce very good results even if these results are not optimal. The quality of an approximation algorithm for the weighted matching problem is measured by its so-called *performance ratio*. An approximation algorithm has a performance ratio of c, if for all graphs it finds a matching with a weight of at least c times the weight of an optimal solution.

Approximation algorithms for the weighted matching problem have been used in practice already for a long time. Their good running times are one of the main motivations for using them. Another reason why these algorithms are used in practice is that they usually require only a few lines of code for their implementation contrary to several thousand lines of code that a good implementation of Edmond's algorithm may require [3].

The two approximation algorithms that are most often used in practice are variants of the maximal matching algorithm and the greedy algorithm. A *maximal matching* in a graph is a matching that is not properly contained in any other matching. Such a matching can easily be computed by starting with an empty matching and extending it in each step by an arbitrary edge in such a way that it remains a matching. Several variants of this simple algorithm are used in practice [14]. The advantage of maximal matching algorithms is that they have linear running time. The major disadvantage of these algorithms is that they have a performance ratio of 0, i.e., the solutions returned by these algorithms can be arbitrarily bad. The greedy algorithm works similarly to the maximal matching algorithm but chooses in each step not an arbitrary but the heaviest edge currently available. It is easy to see that the greedy algorithm has a performance ratio of $\frac{1}{2}$ [1]. The running time of this algorithm is $O(m \log n)$ as it requires sorting the edges of the graph by decreasing weight.

Preis [22] was the first who was able to combine the advantages of the greedy algorithm and the maximal matching algorithm in one algorithm. In 1999 he presented a linear time approximation algorithm for the weighted matching problem with a performance ratio of $\frac{1}{2}$. The idea of his algorithm is to replace the heaviest edge that is needed in the greedy algorithm by a so called *locally heaviest* edge. It is easy to see that the performance ratio of Preis' algorithm is $\frac{1}{2}$. But it is difficult to prove that finding a locally heaviest edge in each step can be done in such a way that the total running time remains linear.

By using a completely different approach Drake and Hougardy [4] obtained another linear time approximation algorithm for the weighted matching problem with a performance ratio of $\frac{1}{2}$. The main idea of their algorithm is to grow in a greedy way *two* matchings simultaneously and return the heavier of both as the result. Their algorithm and its analysis are simpler than that of Preis.

In [5] the idea of local improvements is used as a postprocessing step to enhance the performance of approximation algorithms for the weighted matching problem in practice. This postprocessing step requires only linear running time and it is shown for a large set of test instances that it significantly improves the quality of the solutions. However, this postprocessing step does not improve the performance ratio of $\frac{1}{2}$.

In this paper we prove the existence of linear time approximation algorithms for the weighted matching problem that have performance ratios arbitrarily close to $\frac{2}{3}$.

Main Theorem. *For each $\varepsilon > 0$ there exists a linear time approximation algorithm for the weighted matching problem with a performance ratio of $\frac{2}{3} - \varepsilon$.*

As we will show in Section 7 the dependence on ε of the running time of these algorithms is quite moderate. Moreover our new algorithm is easy to implement and therefore is of relevance in practice.

The main idea of our algorithm is to start with a maximal matching M and to increase its weight by local changes. These local changes which we call *short augmentations* add in each step at most two new edges to M while up to four edges of M will be removed. A graph can possess up to $\Omega(n^4)$ short augmentations. To achieve linear running time only some part of these can be looked at. For each edge of the maximal matching M our algorithm only looks at all short augmentations that involve the endpoints of this edge. The maximality of M ensures that the short augmentations considered by the algorithm are in some sense spread evenly over the graph.

As the short augmentations are partly overlapping it can happen that after performing one short augmentation several others are no longer available. For the performance ratio it is therefore important to be able to reject short augmentations that achieve only minor improvements in the weight of the matching. This is achieved by taking only short augmentations into consideration that gain at least some constant factor β. Such augmentations will be called β-augmentations. In linear time it seems not to be possible to find the best β-augmentation. However we will show that in linear time a constant factor approximation of the best β-augmentation can be found.

To prove the performance ratio of our algorithm we use an amortized analysis. The idea is that the gain that is achieved by an augmentation is not realized immediately but part of it is stored for later use. This way we are able to prove that the algorithm increases the weight of the given matching by some constant factor. By repeating the algorithm a constant number of times and choosing β sufficiently small the resulting matching will have a weight that comes arbitrarily close to $\frac{2}{3}$.

This paper is organized as follows. In Section 2 we give basic definitions and define in Section 3 short augmentations which are the key concept for our algorithm. In Section 4 we present a linear time algorithm that improves the weight of a given matching by a constant factor. The existence of such an algorithm allows us to prove the Main Theorem. We analyze this algorithm in Section 5. A subroutine that finds a constant factor approximation of a best β-augmentation is presented in Section 6. Finally, we prove in Section 7 that the running time of our algorithm depends linearly on $1/\varepsilon$. We conclude in Section 8 with some remarks on the performance of our algorithm in practice.

2 Preliminaries

For an edge $e = \{x, y\}$ we call x and y the *end vertices* of the edge e. Two different edges are *adjacent* if they have a vertex in common. Let $G = (V, E)$ be a weighted graph with a weight function $w : E \to \mathbb{R}^+$. The *weight* of a matching $M \subseteq E$ is defined as $w(M) := \sum_{e \in M} w(e)$. A matching $M \subseteq E$ is called *maximal* if no proper superset of M in E is a matching. By M_{opt} we denote a maximum weight matching in G, i.e., a matching that satisfies $w(M_{opt}) \geq w(M)$ for all other matchings $M \subseteq E$.

A path or cycle is called *M-alternating* if it uses alternately edges from M and $E \setminus M$. The *length* of a path or cycle is the number of edges it contains. Alternating cycles must have even length. For a set F of edges we denote by $M(F)$ all edges in M that are incident with an end vertex of an edge in F. Note that if F contains edges of M then these edges are also contained in $M(F)$.

3 Short Augmentations

Let M be a matching in a graph $G = (V, E)$. If the weight of M is not largest possible then one can replace some set of edges of M by some other set of edges of E such that the new set thus obtained is again a matching and has strictly larger weight than M. We call a process which removes the set $M(S)$ and adds a set $S \subset E$ to M an *augmentation.* The set S is called the *augmenting set* for this augmentation. For technical reasons we allow to add edges to M that are already contained in M. The *gain* of an augmentation with augmenting set S is defined as $w(S) - w(M(S))$ which is the increase of weight it achieves. A *short augmentation* is an augmentation such that all edges in the augmenting set are adjacent to some edge $e \in E$. Such an edge e is called a *center* of this short augmentation. In Figure 1 examples of short augmentations are shown. The examples shown on the left are all possibilities where the center e belongs to M. On the right some examples of short augmentations are shown where the center e does not belong to M. A short augmentation may have more than one center. If this is the case then we assume in the following that one of these edges is chosen to be the center.

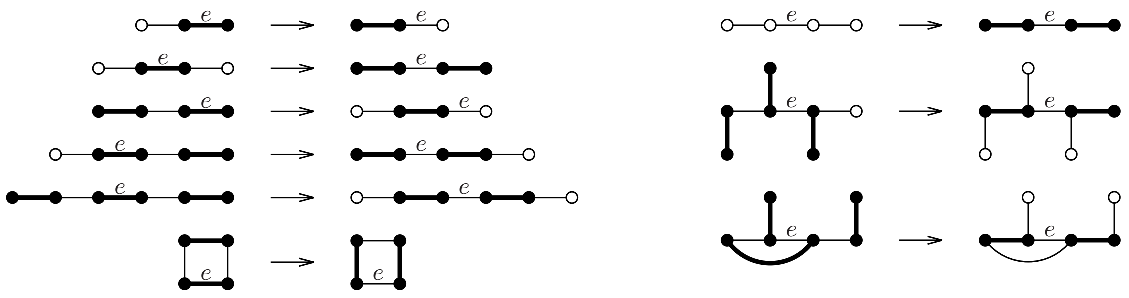

Fig. 1. Examples of short augmentations. A center of each augmentation is labelled e. Edges belonging to the matching are shown in bold. Hollow vertices are vertices not contained in any matching edge.

4 The Algorithm

The main idea of our algorithm is to start with a maximal matching M and to increase the weight of M by performing short augmentations which have as their centers edges of M. A short augmentation might increase the weight of M by a small amount while destroying several other potential short augmentations which could have yielded a much larger increase in weight. An example of such a situation is shown in Figure 2a). The matching M contains the edges of weight 1 and ε. The short augmentation having the edge of weight 1 as its center simply replaces this edge by the neighboring edge of weight $1 + \varepsilon$. After performing this short augmentation there is no short augmentation that has the edge of weight ε as its center. Thus, the total weight of the matching has increased by only ε which may be arbitrarily small.

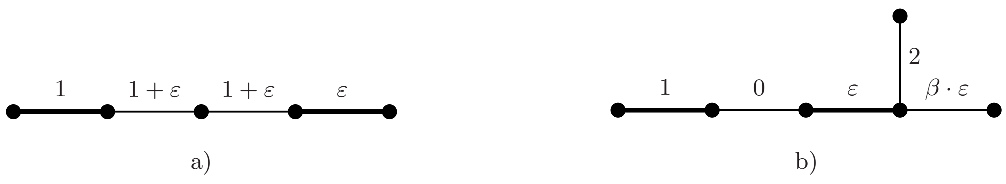

Fig. 2. Examples motivating the use of β-augmentations. Edges belonging to the matching are shown in bold.

To avoid such situations we perform only *β-augmentations*. For a constant $\beta > 1$ a β-augmentation is a short augmentation such that the augmenting set S has the property that $w(S) \geq \beta \cdot w(M(S))$.

The example in Figure 2b) shows that it is not enough to perform any β-augmentation. Instead a β-augmentation having the largest possible gain should be selected. A β-augmentation with center e that has the largest possible gain from among all β-augmentations with center e will be called the *best* β-augmentation with center e. To achieve linear running time our algorithm will not select best β-augmentations but *good* β-augmentations. A β-augmentation with center e is called *good* if it achieves at least a $(\beta - 1)/(\beta - \frac{1}{2})$ fraction of the gain that a best β-augmentation with center e can achieve. In Section 6 we will show that good β-augmentations can be found sufficiently fast.

```
Algorithm improve_matching ($G = (V, E), w : E \to \mathbb{R}^+$, M)
1  make $M$ maximal
2  $M' := M$
3  for $e \in M$ do begin
4    if there exists a $\beta$-augmentation in $M'$ with center $e$
5      then augment $M'$ by a good $\beta$-augmentation with center $e$
6  end
7  return $M'$
```

Fig. 3. Algorithm improve_matching for increasing the weight of a matching.

Our algorithm for finding a matching of weight arbitrarily close to $\frac{2}{3} \cdot w(M_{opt})$ iteratively applies the algorithm improve_matching which is shown in Figure 3. After making the input matching M maximal no further changes will be made to M by the algorithm. Instead M

is copied to M' and all augmentations done in the following are performed with respect to M'. The algorithm visits each edge $e \in M$ exactly once and determines if there is any β-augmenting set centered at this edge in M'. If this is the case then the algorithm performs a good β-augmentation centered at e in M'.

Theorem 1. *If the algorithm improve_matching takes a matching M as input then it returns in linear time a matching M' such that*

$$w(M') \;\geq\; w(M) + \frac{\beta-1}{2\beta}\left(\frac{2}{3\beta}\cdot w(M_{opt}) - w(M)\right) .$$

Before proving this theorem in Section 5 we will show here how Theorem 1 implies the Main Theorem.

Proof of the Main Theorem: Let M_0 be a matching of weight at least $\frac{1}{2}\cdot w(M_{opt})$. Such a matching can be found in linear time [22, 4]. Now apply the algorithm improve_matching to the matching M_i to obtain a matching M_{i+1} for $i = 0, \ldots, k-1$. Then we have $w(M_{i+1}) \geq w_{i+1} \cdot w(M_{opt})$ where w_{i+1} is defined by the following recurrence

$$w_{i+1} \;=\; w_i + \frac{\beta-1}{2\beta}\left(\frac{2}{3\beta} - w_i\right), \text{ and } \quad w_0 = \frac{1}{2} .$$

By solving this linear recurrence equation (see for example [23]) we get

$$w(M_k) \;\geq\; \left(\frac{2}{3\beta} + \left(\frac{1}{2} - \frac{2}{3\beta}\right)\left(\frac{1}{2} + \frac{1}{2\beta}\right)^k\right)\cdot w(M_{opt}) .$$

This shows that for any fixed β and sufficiently large k (depending only on β) there exists a linear time algorithm which finds a matching of weight arbitrarily close to $\frac{2}{3\beta}\cdot w(M_{opt})$. As β can be chosen arbitrarily close to 1 this proves that a matching of weight at least $\left(\frac{2}{3} - \varepsilon\right)\cdot w(M_{opt})$ can be found in linear time. $\square$

In Section 7 we will more carefully analyse how the number of iterations required to achieve a matching of weight at least $\left(\frac{2}{3} - \varepsilon\right)\cdot w(M_{opt})$ depends on ε.

5 Analysis of the Algorithm

In this section we prove Theorem 1. In line 1 the algorithm improve_matching adds edges from E to M until M is maximal. Because this is easy to do we assume from now on that the input matching M is maximal.

Next we will define a set $C \subseteq M$ of centers of short augmentations that we use for the analysis. Consider the symmetric difference $M \triangle M_{opt}$. It consists of M-alternating paths and of even length M-alternating cycles. From each cycle in $M \triangle M_{opt}$ of length larger than four put all edges of this cycle that belong to M into C. For each cycle of length four put exactly one of the two edges in the cycle that belong to M into C. For the M-alternating paths number the edges of M_{opt} in the order in which they appear in these paths. Now partition the edges of M_{opt} in the M-alternating paths into three sets by taking the edge numbers modulo 3. Removing any of these three sets from $M \triangle M_{opt}$ will split the M-alternating paths into short augmentations with centers in M. Removing the lightest of these three sets shows that $2/3$ of the weight of edges of M_{opt} that are contained in M-alternating paths can be achieved by a node disjoint set of short augmentations each of which has a center in M. For each such short augmentation we put a center in C.

For an edge $e \in C$ denote by S_e the augmenting set consisting of the at most two edges of M_{opt} adjacent with e. The gain of such an augmentation is $w(S_e) - w(M(S_e))$. For the cycles of length at least six in $M \triangle M_{opt}$ each edge of M_{opt} belongs to exactly two sets S_e and each edge in M belongs to exactly three sets $M(S_e)$ with $e \in C$. For the cycles of length four and paths in $M \triangle M_{opt}$ each edge of M belongs to exactly one set $M(S_e)$ and the edges in M_{opt} belong to at most one set S_e with $e \in C$. The idea of our analysis is to show that when our algorithm visits an edge $e \in C$ it finds a short augmentation resulting in an *amortized* gain which is at least a constant fraction of $\frac{1}{\beta} w(S_e) - w(M(S_e))$.

The algorithm improve_matching selects good β-augmentations for each $e \in M$ with respect to the matching M' which will be changed during the algorithm. For the analysis we compare for each $e \in C$ the gain that the algorithm finds at e in M' with the value $w(S_e) - w(M(S_e))$, i.e. the gain that could have been achieved if edge e was the first edge considered by the algorithm. Denote by ALG_e the augmenting set that the algorithm improve_matching chooses as a good β-augmentation in M' with center e. The gain that is achieved by this augmentation is $w(ALG_e) - w(M'(ALG_e))$.

The basis for the amortized analysis is the following observation. As the algorithm only performs β-augmentations we know that $w(ALG_e) \geq \beta \cdot w(M'(ALG_e))$. Therefore we have

$$w(ALG_e) - w(M'(ALG_e)) \;\geq\; w(ALG_e) - \frac{1}{\beta} \cdot w(ALG_e) \;=\; \frac{\beta - 1}{\beta} \cdot w(ALG_e) \;.$$

This means if there is a *new* edge f in M', which we define to be an edge $f \in M'$ such that $f \notin M$, then the weight of the matching has already increased at some time in the past by at least $\frac{\beta-1}{\beta} \cdot w(f)$ all of which can be attributed to this one edge independently of any other such new edges.

For the amortized analysis we keep track of two non-negative values, $\overline{w}(M')$ and M''s *savings* where $w(M')$ is the sum of $\overline{w}(M')$ and M''s savings. Deposits to savings and withdrawals from savings are made with respect to certain new edges in amounts that are proportional to the weights of these new edges.

We define a function $\gamma : C \to \{1, 2\}$ by $\gamma(e) = 2$ if e belongs to a cycle of length at least six in $M \triangle M_{opt}$. For all other $e \in C$ the value of $\gamma(e)$ is equal to 1. The idea of the function γ is to count how many short augmentations contain an edge of M_{opt}. When the algorithm visits $e \in M$ in the current matching M' we apply the following rules for the amortized analysis.

Rules for the amortization

1. If $e \notin C$ put all of the gain of the augmentation found at e in savings. This is at least $\frac{\beta-1}{\beta}$ times the weight of the edges added to M' in this step.
2. If $e \in C$ then put $1 - \frac{1}{2\gamma(e)}$ of the gain of the augmentation found at e in savings and increase $\overline{w}(M')$ by the remainder. This results in savings being increased by at least $\left(1 - \frac{1}{2\gamma(e)}\right) \cdot \frac{\beta-1}{\beta}$ times the sum of the weights of any new edges added to M' in this step.
3. If $e \in C$ and if $w(M'(S_e)) > w(M(S_e))$ then withdraw $\frac{\beta-1}{2\gamma(e)\beta}(w(M'(S_e)) - w(M(S_e)))$ from savings and add it to $\overline{w}(M')$.

Rules 1 and 2 increase M''s savings and guarantee that savings will never be overdrawn when Rule 3 is invoked. To see this first note that Rule 3 withdraws upon the new edges $M'(S_e) \setminus M(S_e)$ an amount of $\frac{\beta-1}{2\gamma(e)\beta}(w(M'(S_e)) - w(M(S_e))) \leq \frac{\beta-1}{\beta}(w(M'(S_e) \setminus M(S_e)))$. For a new edge $f \in M'(S_e) \setminus M(S_e)$ the increase of weight to M that can be attributed to f is $\frac{\beta-1}{\beta}w(f)$. We distinguish three cases. If the edge f connects two vertices that each belong to a cycle of length at least six in $M \triangle M_{opt}$ then it can be incident with at most four different short augmentations. One of these short augmentations may result in f becoming a new member of M' and Rule 1 or 2 is applied with $\gamma = 2$. Therefore M''s savings are increased by at least $\frac{3}{4}\frac{\beta-1}{\beta}w(f)$. Afterwards at most three applications of Rule 3 can occur to f each withdrawing $\frac{1}{4}\frac{\beta-1}{\beta}w(f)$. If the edge f connects two vertices where exactly one of these belongs to a cycle of length at least six in $M \triangle M_{opt}$ then it can be incident with at most three different short augmentations which withdraw from M''s savings due to f. If the edge f becomes a new member of M' by Rule 1 then at most $2 \cdot \frac{1}{4} + \frac{1}{2} = 1$ times $\frac{\beta-1}{\beta}w(f)$ will be withdrawn in the following steps. The equation $2 \cdot \frac{1}{4} + \frac{1}{2} - \frac{1}{2\gamma} = 1 - \frac{1}{2\gamma}$ shows that if f becomes a new member of M' by Rule 1 then M''s savings will not be overdrawn by Rule 3. The right hand side of this equation times $\frac{\beta-1}{\beta}w(f)$ is the amount that will be put to M''s savings by Rule 2 with $\gamma \in \{1, 2\}$. The left hand side times $\frac{\beta-1}{\beta}w(f)$ is the amount that at most will be withdrawn from M''s savings by Rule 3. For the third case, where the edge f connects two vertices that both do not belong to a cycle of length at least six in $M \triangle M_{opt}$ there can be at most two short augmentations which withdraw from M''s savings due to f. If f becomes a new member of M' by Rule 1 then at most $2 \cdot \frac{1}{2} = 1$ times $\frac{\beta-1}{\beta}w(f)$ will be withdrawn. Otherwise, if Rule 2 was applied then M''s savings increase by $\frac{1}{2}\frac{\beta-1}{\beta}w(f)$ and at most $\frac{1}{2}\frac{\beta-1}{\beta}w(f)$ will be withdrawn.

Lemma 1. *Amortized Rules 1,2, and 3 guarantee that for every $e \in C$ the algorithm improve_matching achieves an amortized gain of $\frac{\beta-1}{2\gamma(e)\beta}(\frac{1}{\beta} \cdot w(S_e) - w(M(S_e)))$.*

Proof. For proving the Lemma 1 we only have to consider $e \in C$ with $\frac{1}{\beta}w(S_e) - w(M(S_e)) > 0$. As the algorithm performs only good β-augmentations, it finds at least a $\delta := (\beta - 1)/(\beta - \frac{1}{2})$ fraction of the largest possible gain of a β-augmentation with center e in M'. There are three possibilities for the set S_e when the algorithm visits $e \in C$.

The first possibility for S_e is that $w(M'(S_e)) \leq w(M(S_e))$ and therefore S_e is β-augmenting in M' since $w(S_e) \geq \beta \cdot w(M(S_e)) \geq \beta \cdot w(M'(S_e))$. Because $e \in C$ one applies Rule 2 and puts $1 - \frac{1}{2\gamma(e)}$ of the gain of the augmentation at e in savings and increases $\overline{w}(M')$ by

$$\begin{aligned}\frac{1}{2\gamma(e)} \cdot (w(ALG_e) - w(M'(ALG_e))) &\geq \frac{\delta}{2\gamma(e)} \cdot (w(S_e) - w(M'(S_e)))\\ &\geq \frac{\delta}{2\gamma(e)} \cdot (w(S_e) - w(M(S_e)))\\ &\geq \frac{\delta}{2\gamma(e)} \cdot (\frac{1}{\beta} \cdot w(S_e) - w(M(S_e))).\end{aligned}$$

The second possibility is that $w(M'(S_e)) > w(M(S_e))$ and $w(S_e) \geq \beta \cdot w(M'(S_e))$, i.e., the set S_e is still β-augmenting. According to Rule 3 we withdraw for the new edges $M'(S_e) \setminus M(S_e)$ an amount of $\frac{\beta-1}{2\gamma(e)\beta} \cdot (w(M'(S_e)) - w(M(S_e)))$ from M''s savings. This together with the augmentation that the algorithm will find at e, $\frac{1}{2\gamma(e)}$ of which is used to increase $\overline{w}(M')$ according to Rule 2, means that $\overline{w}(M')$ increases by at least

$$\begin{aligned}&\frac{1}{2\gamma(e)}(w(ALG_e) - w(M'(ALG_e))) + \frac{\beta-1}{2\gamma(e)\beta} \cdot (w(M'(S_e)) - w(M(S_e)))\\ &\geq \frac{\delta}{2\gamma(e)}(w(S_e) - w(M'(S_e))) + \frac{\beta-1}{2\gamma(e)\beta}(w(M'(S_e)) - w(M(S_e)))\\ &\geq \min\left(\frac{\delta}{2\gamma(e)}, \frac{\beta-1}{2\gamma(e)\beta}\right)(w(S_e) - w(M'(S_e)) + w(M'(S_e)) - w(M(S_e)))\\ &\geq \min\left(\frac{\delta}{2\gamma(e)}, \frac{\beta-1}{2\gamma(e)\beta}\right)(\frac{1}{\beta}w(S_e) - w(M(S_e))).\end{aligned}$$

The third and final possibility for $e \in C$ is that S_e is no longer β-augmenting when the algorithm visits it, i.e., $w(M'(S_e)) > \frac{1}{\beta} \cdot w(S_e)$. Therefore, the set of edges $M'(S_e) \setminus M(S_e)$ has a weight of at least $\frac{1}{\beta} \cdot w(S_e) - w(M(S_e))$. According to Rule 3 the value $\overline{w}(M')$ increases in this step by at least

$$\frac{\beta-1}{2\gamma(e)\beta}(\frac{1}{\beta}w(S_e) - w(M(S_e))).$$

The minimum amount that $\overline{w}(M')$ increases by at each $e \in C$ over all three cases is $\frac{\beta-1}{2\gamma(e)\beta}(\frac{1}{\beta}w(S_e) - w(M(S_e)))$ as δ was defined as $(\beta-1)/(\beta-\frac{1}{2})$. □

Lemma 1 now easily allows us to prove Theorem 1.

Proof of Theorem 1. Lemma 1 shows that the total gain achieved by the algorithm improve_matching is at least

$$\sum_{e\in C} \frac{\beta-1}{2\gamma(e)\beta}\left(\frac{1}{\beta}\cdot w(S_e) - w(M(S_e))\right) .$$

Consider the symmetric difference of $M \triangle M_{opt}$. Denote by $\tilde{M}$ and $\tilde{M}_{opt}$ all edges in M and M_{opt} respectively that lie in a cycle of length at least six in $M \triangle M_{opt}$. By $\tilde{C}$ denote the set $C \cap \tilde{M}$. Furthermore observe that for an augmentation that is not β-augmenting we have $\frac{1}{\beta}\cdot w(S_e) - w(M(S_e)) < 0$. Therefore we get

$$\begin{aligned}
&\sum_{e\in C} \frac{\beta-1}{2\gamma(e)\beta}\left(\frac{1}{\beta}\cdot w(S_e) - w(M(S_e))\right) \\
&= \sum_{e\in C\setminus\tilde{C}} \frac{\beta-1}{2\beta}\left(\frac{1}{\beta}\cdot w(S_e) - w(M(S_e))\right) + \sum_{e\in\tilde{C}} \frac{\beta-1}{4\beta}\left(\frac{1}{\beta}\cdot w(S_e) - w(M(S_e))\right) \\
&\geq \frac{\beta-1}{2\beta}\left(\frac{2}{3\beta}w(M_{opt}\setminus\tilde{M}_{opt}) - w(\tilde{M})\right) + \frac{\beta-1}{4\beta}\left(\frac{2}{\beta}w(\tilde{M}_{opt}) - 3\cdot w(M\setminus\tilde{M})\right) \\
&\geq \frac{\beta-1}{2\beta}\left(\frac{2}{3\beta}w(M_{opt}) - w(M)\right) .
\end{aligned}$$

For the linear running time we have to show that good β-augmentations can be found sufficiently fast. This will be done in the next section. □

6 Finding a good β-augmentation

Let $G = (V, E)$ be a graph, $M \subseteq E$ be a matching and $e \in E$. Consider a β-augmentation with center e. The *win* of an edge $a \in E \setminus M$ that is adjacent with e is defined as $win_e(a) := w(a) - w(M(a) \setminus \{e\})$, i.e., the win of a is simply the weight of a minus the weight of all edges different from e in M that are adjacent with a. The idea of our algorithm for finding good β-augmentations is as follows. If the best β-augmenting set contains only one edge then it is easy to see that one can compute this set in the required time. If the best β-augmenting set centered at e contains two edges then we first compute the win of all edges adjacent to e and look for a pair of such edges each incident to a different end vertex of e such that these two edges yield a good β-augmentation. Our algorithm for finding good β-augmentations only considers augmenting sets S such that

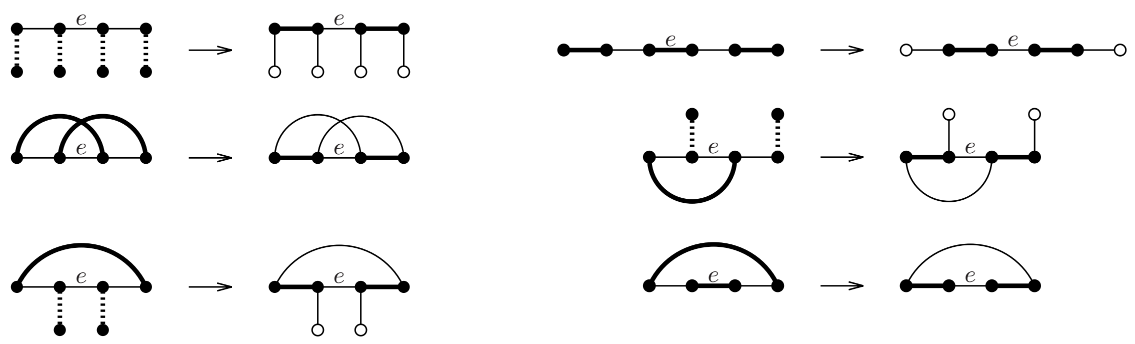

Fig. 4. All types of short augmentations that have two edges in the augmenting set. A center of each augmentation is labelled e. Edges belonging to the matching are shown in bold. Dashed lines indicate edges that are optional. Hollow vertices are vertices not contained in any matching edge.

$S \cap M(S) = \emptyset$. However, it implicitly also considers sets S where this is not true because if S is β-augmenting in M then so is $S \setminus M(S)$ and the gain of this set is just as large.

Lemma 2. *If e is the center of a β-augmentation then a good β-augmentation with center e can be found in time proportional to the sum of the degrees of the end vertices of e.*

Proof. First note that all β-augmentations centered at e in which the augmenting set contains at most one edge can be enumerated and the largest one chosen in the required time. Also β-augmentations in which the augmenting set contains two edges a and b such that $\{a, b, e\} \cup M(a)$ contains a cycle can be enumerated in this time. There are two types of these (see rows 2 and 3 in Fig. 4). The first is that $M(a)$ contains an edge adjacent to e. In this case the edge a is unique and one simply has to enumerate all possibilities for b. Thus a and the best candidate for b can be found in time proportional to the sum of the degrees of the end vertices of e. The second case is that $M(a) \cap M(b)$ contains an edge not incident with an end vertex of e. To enumerate all such sets we scan all edges a incident to one end vertex of e and mark all the end vertex of $M(a)$ which is not incident to a. Then scan all neighbors of the other end vertex of e to see whether any of these are marked.

It therefore remains to consider β-augmentations whose augmenting set contains two edges a and b such that $\{a, b, e\} \cup M(a)$ does not contain a cycle. Define the *surplus* of an edge $a \in E \setminus M$ adjacent with e as $surplus_e(a) := w(a) - \beta \cdot w(M(a) \setminus \{e\})$. We claim that the algorithm good_β_augmentation which is shown in Figure 6 finds a good β-augmentation. This algorithm uses as a subroutine a procedure called max_allowable which returns from all β-augmentations where the augmenting set is contained in F the one containing an edge with the largest win. This algorithm is shown in Figure 5. It simply scans all edges incident with one end vertex of e and checks whether there exists a β-augmentation centered at e that contains this edge. To this end the two largest surpluses achievable on the other side of e are memorized. The reason that we need to consider the

```
Algorithm max_allowable ($G = (V, E), w : E \to \mathbb{R}^+, M, e = \{x, y\} \in E, F \subseteq E$)
1 $F_x := \{f \in F | x \in f\}$, $F_y := \{f \in F | y \in f\}$
2 choose $f_1, f_2 \in F_x$ s.t. $surplus_e(f_1) \geq surplus_e(f_2) \geq surplus_e(f) \ \ \forall f \in F_x \setminus \{f_1, f_2\}$
3 choose $f_3, f_4 \in F_y$ s.t. $surplus_e(f_3) \geq surplus_e(f_4) \geq surplus_e(f) \ \ \forall f \in F_y \setminus \{f_3, f_4\}$
4 if $e \in M$ then $z := w(e)$ else $z := 0$
5 $X_1 := \{f \in F_x | f, f_1$ non adjacent and $surplus_e(f) + surplus_e(f_1) \geq \beta \cdot z\}$
6 $X_2 := \{f \in F_x | f, f_2$ non adjacent and $surplus_e(f) + surplus_e(f_2) \geq \beta \cdot z\}$
7 $X_3 := \{f \in F_y | f, f_3$ non adjacent and $surplus_e(f) + surplus_e(f_3) \geq \beta \cdot z\}$
8 $X_4 := \{f \in F_y | f, f_4$ non adjacent and $surplus_e(f) + surplus_e(f_4) \geq \beta \cdot z\}$
9 return $\beta$-augmentation realizing $max_{f \in X_1 \cup X_2 \cup X_3 \cup X_4} \{win_e(f)\}$
```

Fig. 5. Algorithm max_allowable.

two largest is that one of these may be not disjoint from the other side. Clearly the running time of the algorithm is linear in the sum of the degrees of the end vertices of e.

Let e be the center of a β-augmentation and denote by $gain_{opt}$ the gain of the best β-augmentation with center e. If this β-augmentation contains at most one edge not in M or contains a cycle then it will be found in lines 1 and 2 of the algorithm. Therefore we may assume that the best β-augmentation with center e contains exactly two edges incident with e that do not belong to M.

```
Algorithm good_$\beta$_augmentation ($G = (V, E), w : E \to \mathbb{R}^+, M, e \in E$)
1 $A_1 :=$ best $\beta$-augmentation with center $e$ that contains at most one edge not in $M$
2 $A_2 :=$ best $\beta$-augmentation with center $e$ that contains a cycle
3 $F := \{f \in E \setminus M | f$ is adjacent with $e$ and $win_e(f) \geq \frac{1}{2} \cdot w(e)\}$
4 $A_3 :=$ max_allowable $(F, e)$
5 $F := \{f \in E \setminus M | f$ is adjacent with $e\}$
6 $A_4 :=$ max_allowable $(F, e)$
7 return max $(A_1, A_2, A_3, A_4)$
```

Fig. 6. An algorithm for finding good β-augmentations.

Assume first that both these edges have a win of at least $\frac{1}{2} \cdot w(e)$. Then in line 4 of the algorithm good_β_augmentation a β-augmentation will be returned. Let a and b the two edges in this β-augmentation that do not belong to M. Then $win_e(a) \geq \frac{1}{2} \cdot w(e)$ and $win_e(b) \geq \frac{1}{2} \cdot w(e)$ holds. Without loss of generality assume $win_e(a) \geq win_e(b)$. Then we have

$$gain_{opt} \ \leq \ 2 \cdot win_e(a) - w(e) \ \leq \ 2 \cdot win_e(a) + 2 \cdot win_e(b) - 2 \cdot w(e) \ = \ 2 \cdot gain_{alg} \ .$$

The same argument also shows that in the case $e \notin M$ we have $gain_{opt} \ \leq \ 2 \cdot gain_{alg}$.

Now assume that at least one edge has a win smaller than $\frac{1}{2} \cdot w(e)$. Any β-augmentation with center e has a gain of at least $(\beta - 1)w(e)$. Therefore we have $gain_{alg} \geq (\beta - 1)w(e)$. As by assumption the best β-augmentation with center e contains an edge with win smaller than $\frac{1}{2} \cdot w(e)$ we have $gain_{opt} \leq gain_{alg} + \frac{1}{2} \cdot w(e)$. Now we have

$$gain_{opt} \;\leq\; gain_{alg} + \frac{1}{2} \cdot w(e) \;\leq\; gain_{alg} + \frac{1}{2} \cdot \frac{gain_{alg}}{\beta - 1} \;=\; \frac{\beta - \frac{1}{2}}{\beta - 1} \cdot gain_{alg} \; .$$

The running time is obviously linear in the sum of the degrees of the end vertices of e. $\square$

7 Running Time

In Section 4 we proved that the sequence of matchings $M_0, M_1, \ldots$ obtained by the algorithm improve_matching satisfies $w(M_i) \geq w_i \cdot w(M_{opt})$ where w_i is defined by the following recurrence

$$w_{i+1} \;=\; w_i + \frac{\beta - 1}{2\beta}\left(\frac{2}{3\beta} - w_i\right), \text{ and } \quad w_0 = \frac{1}{2} \; .$$

From this one can easily compute the number of steps required for given ε and β. However, the number of required iterations can be lowered if the value of β is not constant over all iterations. By allowing β to depend on i one gets the recursion

$$w_{i+1} \;=\; w_i + \frac{\beta_i - 1}{2\beta_i}\left(\frac{2}{3\beta_i} - w_i\right), \text{ and } \quad w_0 = \frac{1}{2} \; .$$

The best choice for β_i can easily be calculated by maximizing the expression

$$w_i + \frac{\beta_i - 1}{2\beta_i}\left(\frac{2}{3\beta_i} - w_i\right) \; .$$

This gives $\beta_i = \frac{4}{2+3w_i}$. From this one gets the recursion

$$w_i \;=\; \frac{1}{48}\left(4 + 9w_{i-1}(4 + w_{i-1})\right), \text{ and } \quad w_0 = \frac{1}{2} \; . \tag{1}$$

Now a simple induction shows that $w_i \geq \frac{2}{3} - \frac{16}{3i}$. This is certainly true for w_1. For w_{i+1} we get

$$w_{i+1} \;\geq\; \frac{1}{48}\left(4 + 9\left(\frac{2}{3} - \frac{16}{3i}\right)\left(4 + \left(\frac{2}{3} - \frac{16}{3i}\right)\right)\right) \;=\; \frac{2}{3} + \frac{16}{3i^2} - \frac{16}{3i} \;\geq\; \frac{2}{3} - \frac{16}{3(i+1)} \; .$$

This shows that the number of iterations required to achieve a performance ratio of $\frac{2}{3} - \varepsilon$ is at most $\frac{16}{3\varepsilon} = O(\frac{1}{\varepsilon})$. The following table contains the precise number of iterations needed for a desired performance ratio as can be calculated from the recursion (1).

# iterations	14	18	23	29	37	48	62	82	112	167	286
ratio	0.55	0.56	0.57	0.58	0.59	0.60	0.61	0.62	0.63	0.64	0.65

8 Conclusion

We have presented a linear time approximation algorithm for the weighted matching problem in graphs with an approximation ratio of $2/3 - \varepsilon$ for arbitrarily small $\varepsilon > 0$. The constants in Theorem 1 can be improved slightly which results in slightly better constants in the running time. However, as this would make our analysis more complicated we did not do it. The analysis in Section 7 shows that the predicted running time of our algorithm is quite low. We have implemented our algorithm to test its behaviour on real world instances. We have chosen the same test set of instances for the weighted matching problem as was chosen in [5]. It turns out that in practice the best choice for β is to simply set its value to 1. With this setting our new algorithm outperforms the other algorithms studied in [5] with respect to the approximation ratio while having similar running time.

Acknowledgement

We are grateful to Valentin Ziegler for pointing out that Lemma 1 in [6] is incorrect.

References

1. D. Avis, A Survey of Heuristics for the Weighted Matching Problem, Networks, Vol. 13 (1983), 475–493.
2. R. Beier, J.F. Sibeyn, A Powerful Heuristic for Telephone Gossiping, Proc. 7th Colloquium on Structural Information and Communication Complexity, Carleton Scientific (2000), 17–35.
3. W. Cook, A. Rohe, Computing minimum-weight perfect matchings, INFORMS Journal on Computing 11 (1999), 138–148.
4. D.E. Drake, S. Hougardy, A Simple Approximation Algorithm for the Weighted Matching Problem, Information Processing Letters 85 (2003), 211–213.
5. D.E. Drake, S. Hougardy, Linear Time Local Improvements for Weighted Matchings in Graphs, In: Workshop on Efficient Algorithms (WEA) 2003, K.Jansen, M.Margraf, M.Mastrolli, J.D.P.Rolim (eds.), LNCS 2647, Springer 2003, 107–119.
6. D.E. Drake, S. Hougardy, Improved linear time approximation algorithms for weighted matchings, In: Approximation, Randomization, and Combinatorial Optimization, (Approx/Random) 2003, S.Arora, K.Jansen, J.D.P.Rolim, A.Sahai (eds.), LNCS 2764, Springer 2003, 14–23.
7. J. Edmonds, Maximum matching and a polyhedron with 0,1-vertices, J. Res. Nat. Bur. Standards 69B (1965), 125–130.
8. C. Frömmel, C. Gille, A. Goede, C. Gröpl, S. Hougardy, T. Nierhoff, R. Preißner, M. Thimm, Accelerating screening of 3D protein data with a graph theoretical approach, Bioinformatics 19 (2003), 2442–2447
9. H.N. Gabow, An efficient implementation of Edmond's algorithm for maximum matching on graphs, Journal of the ACM 23 (1976), 221–234.
10. H.N. Gabow, Data Structures for Weighted Matching and Nearest Common Ancestors with Linking, SODA 1990, 434–443.
11. H.N. Gabow, Z. Galil, T.H. Spencer, Efficient implementation of graph algorithms using contraction, J. ACM 36 (1989), 815–853.
12. Z. Galil, S. Micali, H. Gabow, An O(EV log V) algorithm for finding a maximal weighted matching in general graphs, SIAM Journal on Computing 15 (1986), 120–130.

13. H.N. Gabow, R.E. Tarjan, Faster Scaling Algorithms for General Graph-Matching Problems, J. ACM 38:4 (1991), 815–853.
14. G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20:1 (1998), 359–392.
15. E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.
16. R.J. Lipton, R.E. Tarjan, A separator theorem for planar graphs, SIAM J. Appl. Math. 36 (1979), 177–189.
17. R.J. Lipton, R.E. Tarjan, Applications of a planar separator theorem, SIAM Journal on Computing 9 (1980), 615–627.
18. R.H. Möhring, M. Müller-Hannemann, Complexity and Modeling Aspects of Mesh Refinement into Quadrilaterals, Algorithmica 26 (2000), 148–171.
19. B. Monien, R. Preis, R. Diekmann, Quality Matching and Local Improvement for Multilevel Graph-Partitioning, Parallel Computing 26(12), 2000, 1609–1634.
20. K. Mehlhorn, G. Schäfer, Implementation of $O(nmlogn)$ Weighted Matchings in General Graphs: The Power of Data Structures, In: S. Näher, D. Wagner (eds.), 4th International Workshop on Algorithm Engineering (WAE) 2000, Saarbrücken, Germany, September 5–8, LNCS 1982, Springer 2001, 23–38.
21. S. Micali and V.V. Vazirani, An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science (1980) 17–27.
22. R. Preis, Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs, Symposium on Theoretical Aspects of Computer Science (STACS) 1999, C. Meinel, S. Tison (eds.), LNCS 1563, Springer 1999, 259–269.
23. P.W. Purdom, C.A. Brown, The Analysis of Algorithms, Holt, Rinehart and Winston, New York, 1985.
24. R.E. Tarjan, Data Structures and Network Algorithms, CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 44, SIAM 1983.
25. V.V. Vazirani, A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ Maximum Matching Algorithm, Combinatorica 14:1 (1994), 71–109.